UM1860
User manual

Getting started with STM32CubeL4 for STM32L4 Series

Introduction

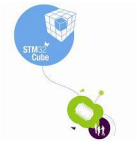
STMCube™ is an STMicroelectronics original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube is the implementation of STMCube™ that covers STM32 microcontrollers.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 series):
  - The STM32Cube HAL, an STM32 abstraction layer embedded software ensuring maximized portability across STM32 portfolio. The HAL is available for all peripherals
  - The Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB, STMTouch™, FatFS and Graphics
  - All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeL4 firmware package.

Section 1 describes the main features of STM32CubeL4 firmware, part of the STMCube™ initiative. *Section 2* and *Section 3* provide an overview of the STM32CubeL4 architecture and firmware package structure.



Contents

1 STM32CubeL4 main features 6

2 STM32CubeL4 architecture overview 8

2.1 Level 0 8

2.1.1 Board Support Package (BSP) 9

2.1.2 Hardware Abstraction Layer (HAL) and Low Layer (LL) 9

2.1.3 Basic peripheral usage examples 9

2.2 Level 1 10

2.2.1 Middleware components 10

2.2.2 Examples based on the middleware components 11

2.3 Level 2 12

3 STM32CubeL4 firmware package overview 13

3.1 Supported STM32L4 devices and hardware 13

3.2 Firmware package overview 14

4 Getting started with STM32CubeL4 17

4.1 Running your first example 17

4.2 Developing your own application 18

4.3 Using STM32CubeMX to generate initialization C code 20

4.4 Getting STM32CubeL4 release updates 21

4.4.1 Installing and running the STM32CubeUpdater program 21

5 FAQ 22

5.1 What is the license scheme for the STM32CubeL4 firmware? 22

5.2 What boards are supported by the STM32CubeL4 firmware package? . 22

5.3 Are any examples provided with the ready-to-use toolset projects? 22

5.4 Is there any link with Standard Peripheral Libraries? 22

5.5 Does the HAL drivers take benefit from interrupts or DMA? How can this be controlled? 22

5.6 How are the product/peripheral specific features managed? 22

5.7 How can STM32CubeMX generate code based on embedded software? 23

5.8 How can I get regular updates on the latest STM32CubeL4 firmware releases? 23
5.9 When should I use HAL versus LL drivers? 23
5.10 How can I include LL drivers in my environment? Is there any LL configuration file as for HAL? 23
5.11 Can I use HAL and LL drivers together? If yes, what are the constraints? 23
5.12 Is there any LL APIs which are not available with HAL 24
5.13 Why are SysTick interrupts not enabled on LL drivers? 24

6 Revision history 25

List of tables

Table 1. Macros for STM32L4 series 13
Table 2. Boards for STM32L4 series 13
Table 3. Number of examples available for each board 16
Table 4. Document revision history 25

List of figures

Figure 1. STM32CubeL4 firmware components 7
Figure 2. STM32CubeL4 firmware architecture 8
Figure 3. STM32CubeL4 firmware package structure 14
Figure 4. STM32CubeL4 examples overview 15

1 STM32CubeL4 main features

STM32CubeL4 gathers, in a single package, all the generic embedded software components required to develop an application on STM32L4 microcontrollers. In line with the STMCube™ initiative, this set of components is highly portable, not only within STM32L4 series but also to other STM32 series.

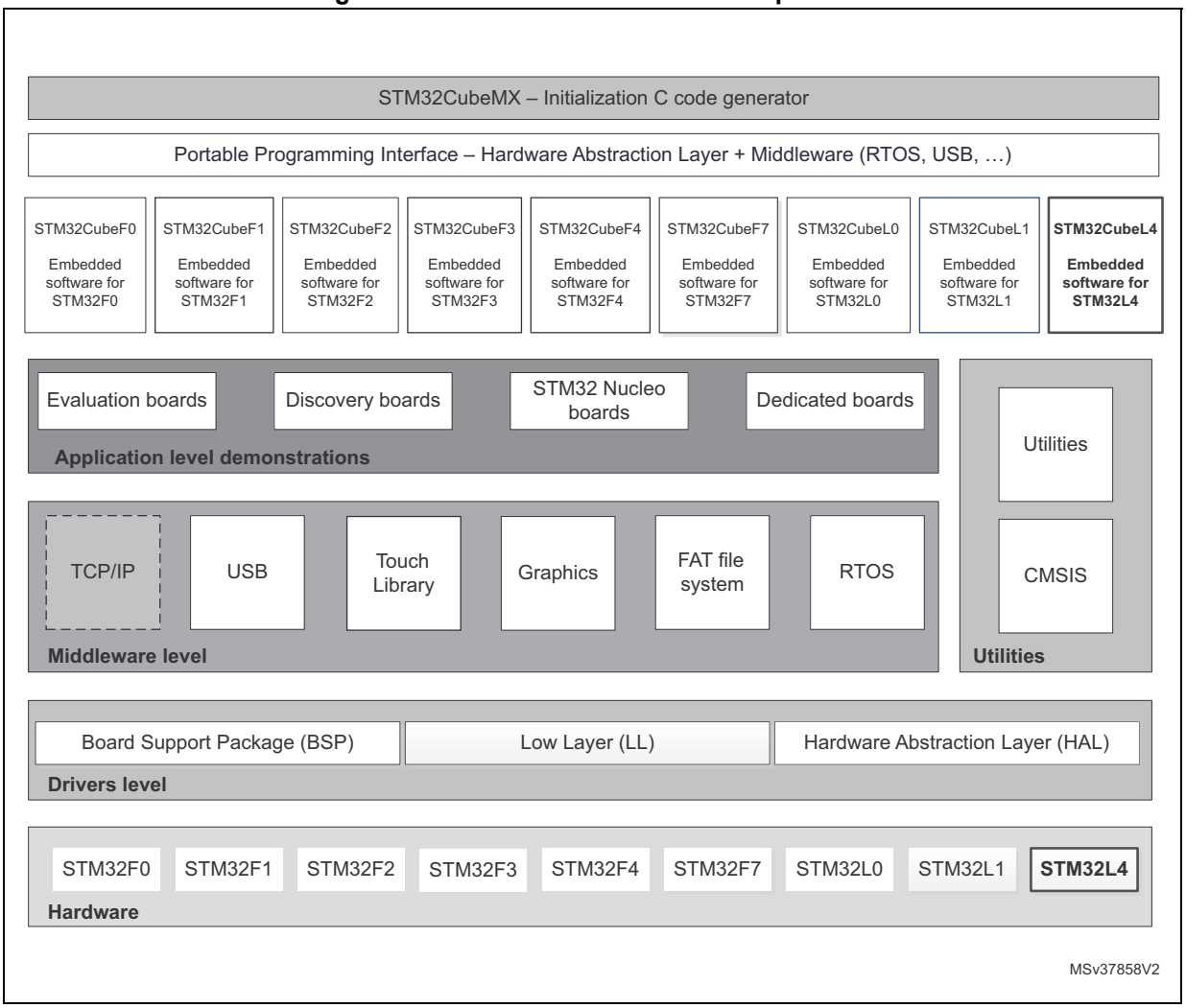
STM32CubeL4 is fully compatible with STM32CubeMX code generator that allows generating initialization code. The package includes Low Layer (LL) and hardware abstraction layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in open-source BSD license for user convenience.

STM32CubeL4 package also contains a set of middleware components with the corresponding examples. They come in free user-friendly license terms:

- Full USB Host and Device stack supporting many classes.
  - Host Classes: HID, MSC, CDC, Audio, MTP
  - Device Classes: HID, MSC, CDC, Audio, DFU, LPM, BCD.
- STemWin, a professional graphical stack solution available in binary format and based on STMicroelectronics partner solution SEGGER emWin
- CMSIS-RTOS implementation with FreeRTOS open source solution
- FAT File system based on open source FatFS solution
- STMTouch touch sensing library solution.

Several applications and demonstrations implementing all these middleware components are also provided in the STM32CubeL4 package.

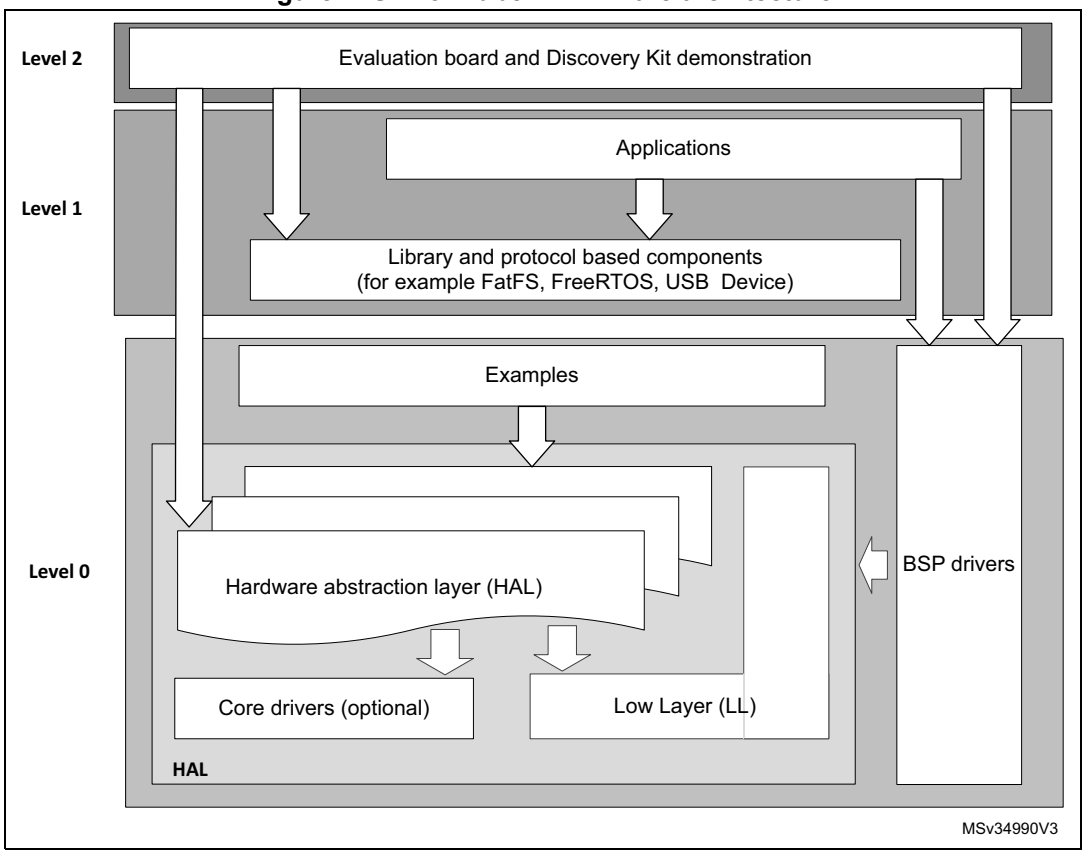
Figure 1. STM32CubeL4 firmware components



2 STM32CubeL4 architecture overview

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other's as described in *Figure 2*:

Figure 2. STM32CubeL4 firmware architecture



2.1 Level 0

This level is divided into three sub-layers:

- Board Support Package (BSP)
- Hardware Abstraction Layer (HAL)
  - HAL peripheral drivers
  - Low Layer drivers
- Basic peripheral usage examples.

2.1.1 Board Support Package (BSP)

This layer offers a set of APIs relative to the hardware components in the hardware boards (such as LCD, Audio, microSD, MEMS drivers). It is composed of two parts:

- Component

  This is the driver relative to the external device on the board and not to the STM32. The component driver provide specific APIs to the BSP driver external components and could be portable on any other board.

- BSP driver

  It allows linking the component driver to a specific board and provides a set of user-friendly APIs. The API naming rule is BSP_FUNCT_Action().

  Example: BSP_LED_Init(), BSP_LED_On()

The BSP is based on a modular architecture allowing an easy porting on any hardware by just implementing the low-level routines.

2.1.2 Hardware Abstraction Layer (HAL) and Low Layer (LL)

The STM32CubeL4 HAL and LL are complementary and cover a wide range of applications requirements:

- The HAL drivers offer high-level function-oriented highly-portable APIs. They hide the MCU and peripheral complexity to end user.

  The HAL drivers provide generic multi-instance feature-oriented APIs which simplify user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART…), it provides APIs allowing initializing and configuring the peripheral, managing data transfer based on polling, interrupt or DMA process, and handling communication errors that may raise during communication. The HAL driver APIs are split in two categories:

  – Generic APIs which provides common and generic functions to all the STM32 series

  – Extension APIs which provides specific and customized functions for a specific family or a specific part number.

- The Low Layer APIs provide low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications.

  The LL drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or for those requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB, or SDMMC).

  They feature:

  – A set of inline functions for direct and atomic register access

  – Full independence from HAL and capability to be used in standalone mode (without HAL drivers)

  – Full coverage of the supported peripheral features.

2.1.3 Basic peripheral usage examples

This layer includes the examples build over the STM32 peripheral and using either the HAL or/and the Low Layer drivers APIs as well as the BSP resources.

2.2 Level 1

This level is divided into two sub-layers:

- Middleware components
- Examples based on the middleware components.

2.2.1 Middleware components

The middleware is a set of libraries covering USB Host and Device Libraries, STMTouch touch sensing, STemWin, FreeRTOS and FatFS. Horizontal interactions between the components of this layer is done directly by calling the feature APIs while the vertical interaction with the low-level drivers is done through specific callbacks and static macros implemented in the library system call interface. For example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

The main features of each middleware component are as follows:

- **USB Host and Device Libraries**
  - Several USB classes supported (Mass-Storage, HID, CDC, DFU, LPM and BCD).
  - Support of multi-packet transfer features that allows sending big amounts of data without splitting them into maximum packet size transfers.
  - Use of configuration files to change the core and the library configuration without changing the library code (Read Only).
  - 32-bit aligned data structures to handle DMA-based transfer in high-speed modes.
  - Support of multi USB OTG core instances from user level through configuration file. This allows to perform operations with more than one USB host/device peripheral.
  - RTOS and Standalone operation.
  - Link with low-level driver through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.
- **STemWin Graphical stack**
  - Professional grade solution for GUI development based on SEGGER's emWin solution.
  - Optimized display drivers.
  - Software tools for code generation and bitmap editing (STemWin Builder…).
- **FreeRTOS**
  - Open source standard.
  - CMSIS compatibility layer.
  - Tickless operation during low-power mode.
  - Integration with all STM32Cube middleware modules.
- **FAT File system**
  - FATFS FAT open source library.
  - Long file name support.
  - Dynamic multi-drive support.
  - RTOS and standalone operation.
  - Examples with microSD.
- **STM32 Touch sensing library**

  Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear and rotary touch sensors. It is based a proven surface charge transfer acquisition principle.

2.2.2 Examples based on the middleware components

Each middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several middleware components are provided as well.

2.3 Level 2

This level is composed of a single layer which consist in a global real-time and graphical demonstration based on the middleware service layer, the low-level abstraction layer and the basic peripheral usage applications for board based features.

3 STM32CubeL4 firmware package overview

3.1 Supported STM32L4 devices and hardware

STM32Cube offers highly portable Hardware Abstraction Layer (HAL) built around a generic architecture. It allows the build-upon layers, such as the middleware layer, to implement their functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability on other devices.

In addition, thanks to its layered architecture, the STM32CubeL4 offers full support of all STM32L4 series. The user has only to define the right macro in stm32l4xx.h.

Table 1 gives the macro to be defined depending on the STM32L4 device used. This macro must also be defined in the compiler preprocessor.

Table 1. Macros for STM32L4 series

Macro defined in stm32l4xx.h	STM32L4 devices
STM32L471xx	STM32L471RG, STM32L471JG, STM32L471VG, STM32L471QG, STM32L471ZG, STM32L471RE, STM32L471JE, STM32L471VE, STM32L471QE, STM32L471ZE
STM32L475xx	STM32L475RG, STM32L475JG, STM32L475VG, STM32L475QG, STM32L475ZG, STM32L475RE, STM32L475JE, STM32L475VE, STM32L475QE, STM32L475ZE, STM32L475RC, STM32L475VC
STM32L476xx	STM32L476RG, STM32L476JG, STM32L476VG, STM32L476QG, STM32L476ZG, STM32L476RE, STM32L476JE, STM32L476VE, STM32L476QE, STM32L476ZE, STM32L476RC, STM32L476VC
STM32L485xx	STM32L485JC, STM32L485JE
STM32L486xx	STM32L486RG, STM32L486JG, STM32L486VG, STM32L486QG, STM32L486ZG

STM32CubeL4 features a rich set of examples and applications at all levels making it easy to understand and use any HAL driver and/or middleware components. These examples are running on STMicroelectronics boards as listed in *Table 2*.

Table 2. Boards for STM32L4 series

Board	STM32L4 devices supported
STM32L476G-EVAL	STM32L476xx
32L476GDISCOVERY	STM32L476xx
NUCLEO-L476RG	STM32L476xx

As for all other STM32 Nucleo boards, the NUCLEO-L476RG feature a reduced set of hardware components (one user key button and one user LED). To enrich the middleware support offer for the STM32CubeL4 firmware package, an LCD display Adafruit Arduino™ shield is used. This shield embeds a microSD connector and a joystick in addition to the LCD.

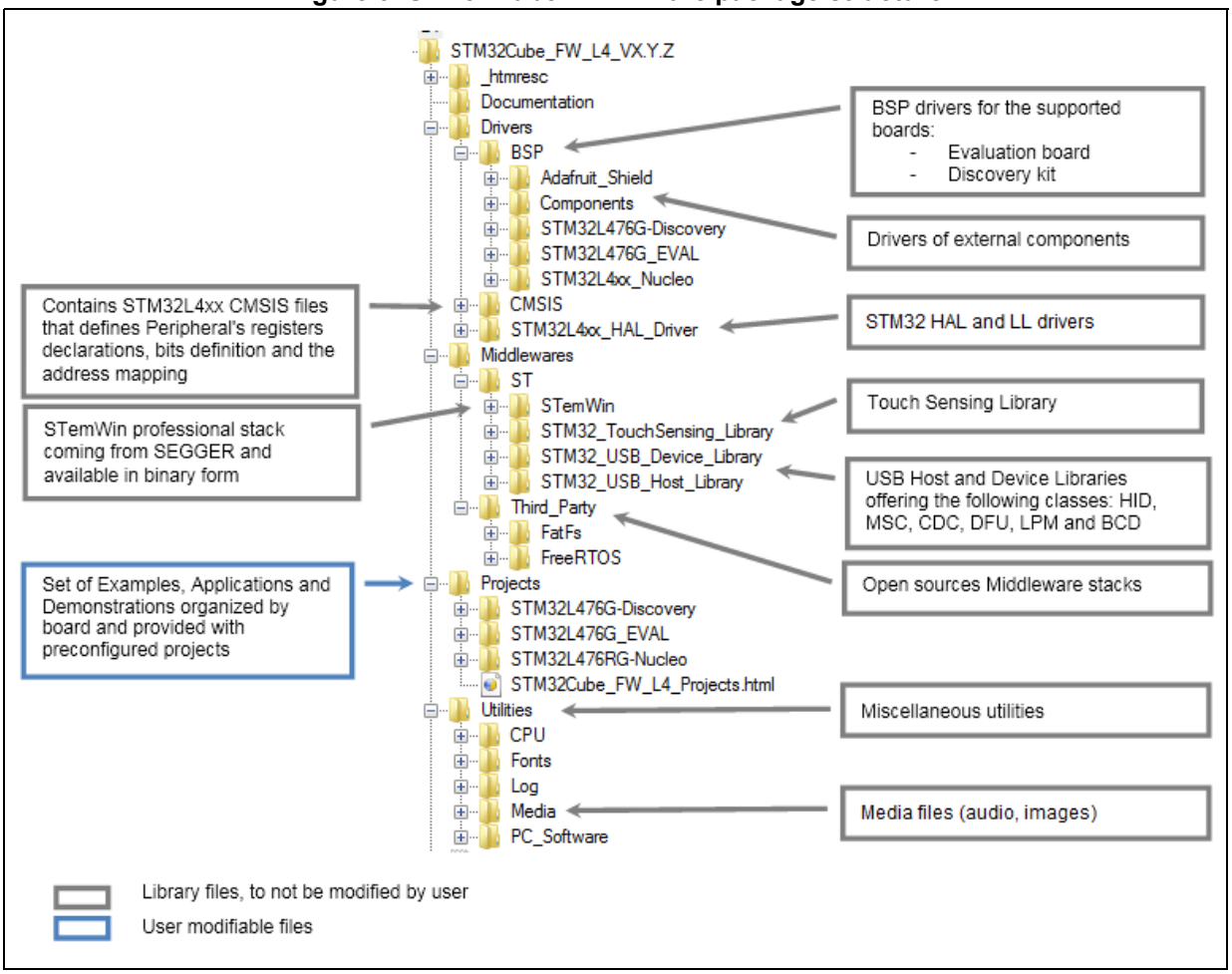
In the BSP component, the dedicated drivers for the Arduino shield are available. Their use is illustrated through either the provided BSP example or in the demonstration firmware, without forgetting the FatFS middleware application.

The STM32CubeL4 firmware is able to run on any compatible hardware. The user simply updates the BSP drivers to port the provided examples on his own board, if this later has the same hardware features (LED, LCD display, buttons...).

3.2 Firmware package overview

The STM32CubeL4 firmware solution is provided in one single zip package having the structure shown in *Figure 3*.

Figure 3. STM32CubeL4 firmware package structure

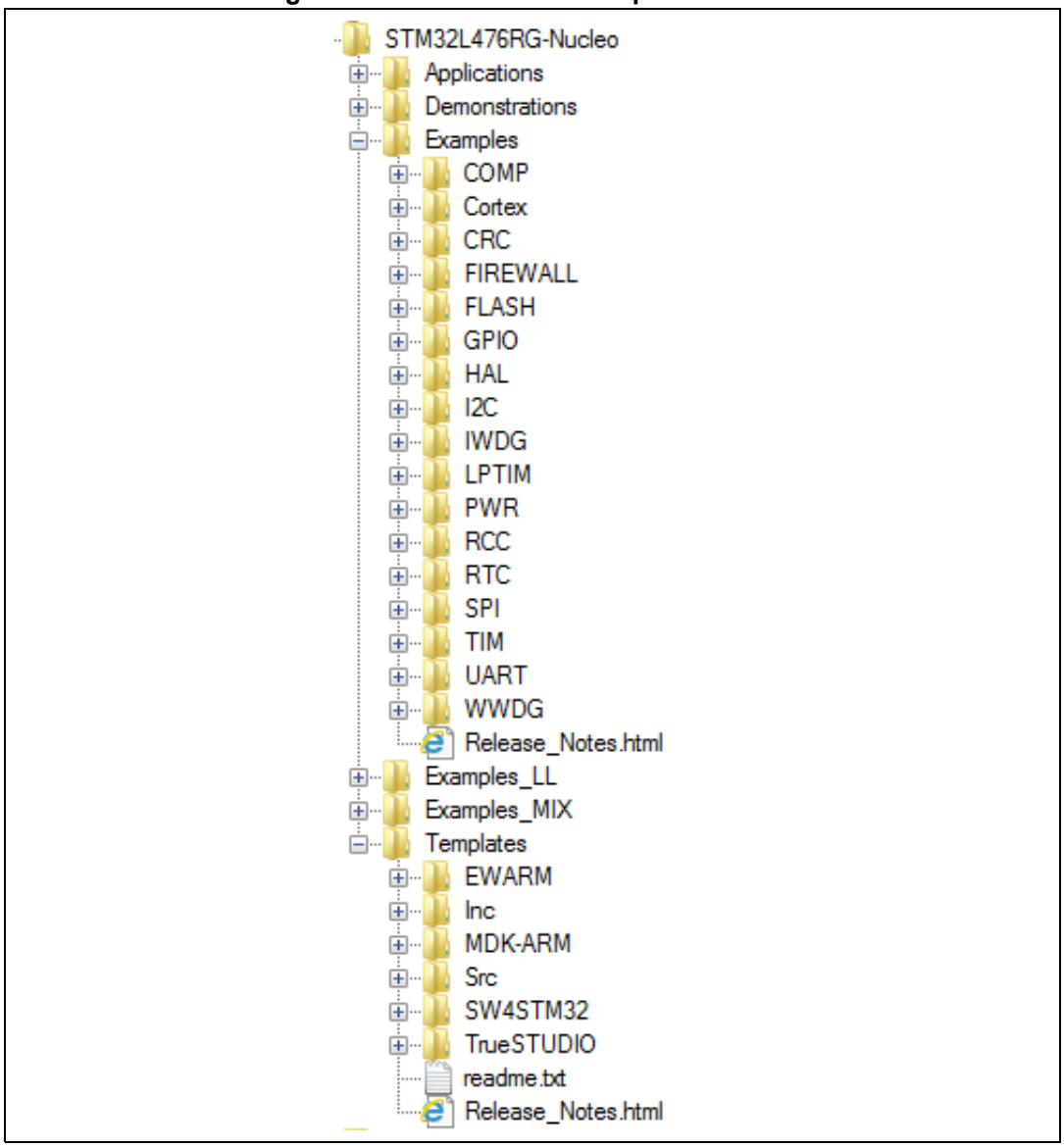


1. The library files in brown must not be modified by the user, while the files in blue can be modified.

For each board, a set of examples are provided with pre-configured projects for EWARM, MDK-ARM, SW4STM32 and TrueSTUDIO toolchains.

Figure 4 shows the projects structure for the NUCLEO-L476RG board.

Figure 4. STM32CubeL4 examples overview



The examples are classified depending on the STM32Cube level they apply to, and are named as explained below:

- Level 0 examples are called *Examples*, *Examples_LL* and *Examples_MIX*. They use respectively HAL drivers, LL drivers and a mix of HAL and LL drivers without any middleware component.
- Level 1 examples are called *Applications.* They provide typical use cases of each middleware component.

The *Template* project available in the Template directory allows to quickly build a firmware application on a given board.

All examples have the same structure:

- *Inc* folder that contains all header files.
- *Src* folder for the sources code.
- *EWARM*, *MDK-ARM*, /*SW4STM32* and *TrueSTUDIO* folders contain the pre-configured project for each toolchain.
- *readme.txt* describing the example behavior and needed environment to make it working

Table 3 gives the number of projects available for each board.

Table 3. Number of examples available for each board

Board	HAL examples	LL examples	Mixed HAL/LL examples	Applications	Demonstration
STM32L476G-EVAL	84	0	0	28	1
32L476GDISCOVERY	16	0	0	1	1
NUCLEO-L476RG	64	72	12	1	1

4 Getting started with STM32CubeL4

4.1 Running your first example

This section explains how simple is to run a first example within STM32CubeL4. It uses as illustration the generation of a simple LED toggle running on STM32L476RG Nucleo board:

1. Download the STM32CubeL4 firmware package. Unzip it into a directory of your choice. Make sure not to modify the package structure shown in *Figure 3*. Note that it is also recommended to copy the package at a location close to your root volume (e.g. C\Eval or G:\Tests) because some IDEs encounter problems when the path length is too long.
2. Browse to \Projects\STM32L476RG-Nucleo\Examples.
3. Open \GPIO, then \GPIO_EXTI folder.
4. Open the project with your preferred toolchain. A quick overview on how to open, build and run an example with the supported toolchains is given below.
5. Rebuild all files and load your image into target memory.
6. Run the example: each time you press the USER pushbutton, the LED2 toggles (for more details, refer to the example readme file).

To open, build and run an example with the supported toolchains:, follow the steps below:

- EWARM
  a) Under the example folder, open \EWARM sub-folder.
  b) Launch the Project.eww workspace[a].
  c) Rebuild all files: **Project->Rebuild all**.
  d) Load project image: **Project->Debug**.
  e) Run program: **Debug->Go(F5)**.
- MDK-ARM
  a) Under the example folder, open \MDK-ARM sub-folder.
  b) Launch the Project.uvprojx workspace[a].
  c) Rebuild all files: **Project->Rebuild all target files**.
  d) Load project image: **Debug->Start/Stop Debug Session**.
  e) Run program: **Debug->Run (F5)**.
- SW4STM32
  a) Open the SW4STM32 toolchain
  b) Click **File->Switch Workspace->Other** and browse to the SW4STM32 workspace directory
  c) Click **File->Import**, select **General->Existing Projects into Workspace** and then click **Next**.
  d) Browse to the SW4STM32 workspace directory and select the project.
  e) Rebuild all project files: select the project in the **Project explorer** window then click the **Project->build project** menu.
  f) Run program: Run->Debug (F11)
- TrueSTUDO
  a) Open the TrueSTUDIO toolchain.
  b) Click **File->Switch Workspace->Other** and browse to TrueSTUDIO workspace directory.
  c) Click **File->Import**, select **General->Existing Projects into Workspace** and then click **Next**.
  d) Browse to the TrueSTUDIO workspace directory, select the project.
  e) Rebuild all project files: select the project in the **Project explorer** window then click the **Project->build project** menu.
  f) Run program: **Run->Debug (F11)**.

4.2 Developing your own application

This section describes the steps required to create your own application using STM32CubeL4:

1. **Create your project**

   To create a new project, you can either start from the *Template* project provided for each board under \Projects\<STM32xxx_yyy>\Templates or from any available project

a. The workspace name may change from one example to another.

under \Projects\<STM32xxy_yyy>\Examples or \Projects\<STM32xx_yyy>\Applications (where <STM32xxx_yyy> refers to the board name, e.g. STM32L476G-EVAL).

The *Template* project is providing empty main loop function, however it's a good starting point to get familiar with project settings for STM32CubeL4. The template has the following characteristics:

– It contains the source code of HAL, CMSIS and BSP drivers which are the minimal components required to develop a code on a given board.
– It contains the include paths for all the firmware components.
– It defines the STM32L4 device supported, thus allowing to configure the CMSIS and HAL drivers accordingly.
– It provides read-to-use user files pre-configured as shown below:

  HAL initialized with default time base with ARM Core SysTick.

  SysTick ISR implemented for HAL_Delay() purpose.

Note: When copying an existing project to another location, make sure to update the include paths.

2. Add the necessary middleware to your project (optional)

The available middleware stacks are: USB Host and Device library, STMTouch touch sensing, STemWin, FreeRTOS and FatFS. To know which source files you need to add in the project files list, refer to the documentation provided for each middleware. You can also look at the Applications available under \Projects\STM32xxx_yyy\Applications\<MW_Stack> (where <MW_Stack> refers to the middleware stack, e.g. USB_Device) to know which sources files and which include paths have to be added.

3. Configure the firmware components

The HAL and middleware components offer a set of build time configuration options using macros # define declared in a header file. A template configuration file is provided within each component, it has to be copied to the project folder (usually the configuration file is named xxx_conf_template.h, the word '_template' needs to be removed when copying it to the project folder). The configuration file provides enough information to know the impact of each configuration option. More detailed information is available in the documentation provided for each component.

4. Start the HAL Library

After jumping to the main program, the application code must call *HAL_Init()* API to initialize the HAL Library, which do the following tasks:

a) Configuration of the Flash prefetch and SysTick interrupt priority (through macros defined in stm32l4xx_hal_conf.h).
b) Configuration of the SysTick to generate an interrupt each 1 msecond at the SysTick interrupt priority TICK_INT_PRIO defined in stm32l4xx_hal_conf.h, which is clocked by the MSI (at this stage, the clock is not yet configured and thus the system is running from the internal 4 MHz MSI).
c) Setting of NVIC Group Priority to 4.
d) Call of HAL_MspInit() callback function defined in stm32l4xx_hal_msp.c user file to perform global low-level hardware initializations.

5. **Configure the system clock**

The system clock configuration is done by calling the two APIs described below:

a) HAL_RCC_OscConfig(): this API configures the internal and/or external oscillators, as well as the PLL source and factors. The user can choose to

configure one oscillator or all oscillators. The PLL configuration can be skipped if there is no need to run the system at high frequency.

b) HAL_RCC_ClockConfig(): this API configures the system clock source, the Flash memory latency and AHB and APB prescalers.

6. **Initialize the peripheral**

   a) First write the peripheral HAL_PPP_MspInit function. Proceed as follows:

   – Enable the peripheral clock.

   – Configure the peripheral GPIOs.

   – Configure DMA channel and enable DMA interrupt (if needed).

   – Enable peripheral interrupt (if needed).

   b) Edit the stm32xxx_it.c to call the required interrupt handlers (peripheral and DMA), if needed.

   c) Write process complete callback functions if you plan to use peripheral interrupt or DMA.

   d) In your main.c file, initialize the peripheral handle structure then call the function HAL_PPP_Init() to initialize your peripheral.

7. **Develop your application**

   At this stage, your system is ready and you can start developing your application code.

   – The HAL provides intuitive and ready-to-use APIs to configure the peripheral. It supports polling, interrupts and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich examples set provided in the STM32CubeL4 package.

   – If your application has some real-time constraints, you can found a large set of examples showing how to use FreeRTOS and integrate it with all middleware stacks provided within STM32CubeL4. This can be a good starting point to develop your application.

Caution: In the default HAL implementation, SysTick timer is used as timebase: it generates interrupts at regular time intervals. If HAL_Delay() is called from peripheral ISR process, make sure that the SysTick interrupt has higher priority (numerically lower) than the peripheral interrupt. Otherwise, the caller ISR process will be blocked. Functions affecting timebase configurations are declared as __weak to make override possible in case of other implementations in user file (using a general purpose timer for example or other time source). For more details, refer to HAL_TimeBase example.

4.3 Using STM32CubeMX to generate initialization C code

An alternative to steps 1 to 6 described in *Section 4.2* consists in using the STM32CubeMX tool to generate code to initialize the system, peripherals and middleware (steps 1 to 6 above) through a step-by-step process:

1. Select the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configure each required embedded software thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU peripheral configuration (e.g. GPIO or USART) and middleware stacks (e.g. USB).
3. Generate the initialization C code based on the configuration selected. This code is ready-to-use within several development environments. The user code is kept at the next code generation.

For more information, please refer to STM32CubeMX user manual (UM1718).

4.4 Getting STM32CubeL4 release updates

The STM32CubeL4 firmware package comes with an updater utility, STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool.

The updater solution detects new firmware releases and patches available from www.st.com and proposes to download them to the user's computer.

4.4.1 Installing and running the STM32CubeUpdater program

Follow the sequence below to install and run the STM32CubeUpdater:

1. To launch the installation, double-click the SetupSTM32CubeUpdater.exe file.
2. Accept the license terms and follow the different installation steps.
3. Upon successful installation, STM32CubeUpdater becomes available as an STMicroelectronics program under *Program Files* and is automatically launched. The STM32CubeUpdater icon appears in the system tray. Right-click the updater icon and select **Updater Settings** to configure the Updater connection and whether to perform manual or automatic checks. For more details on Updater configuration, refer to section 3 of STM32CubeMX user manual - UM1718).

5 FAQ

5.1 What is the license scheme for the STM32CubeL4 firmware?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license.

The middleware stacks made by STMicroelectronics (USB Host and Device Libraries, STemWin) come with a licensing model allowing easy reuse, provided it runs on an STMicroelectronics device.

The middleware based on well-known open-source solutions (FreeRTOS and FatFs) have user-friendly license terms. For more details, refer to the license agreement of each middleware.

5.2 What boards are supported by the STM32CubeL4 firmware package?

The STM32CubeL4 firmware package provides BSP drivers and ready-to-use examples for the following STM32L4 boards: STM32L476G-EVAL, 32L476GDISCOVERY and NUCLEO-L476RG.

5.3 Are any examples provided with the ready-to-use toolset projects?

Yes. STM32CubeL4 provides a rich set of examples and applications. They come with the pre-configured projects for IAR, Keil and GCC toolchains.

5.4 Is there any link with Standard Peripheral Libraries?

The STM32Cube HAL and LL drivers are the replacement of the standard peripheral library:

- The HAL drivers offer a higher abstraction level compared to the standard peripheral APIs. They focus on peripheral common features rather than hardware. Their higher abstraction level allows defining a set of user-friendly APIs that can be easily ported from one product to another.
- The LL drivers offer low-level APIs at registers level.

5.5 Does the HAL drivers take benefit from interrupts or DMA? How can this be controlled?

Yes, they do. The HAL layer supports three API programming models: polling, interrupt and DMA (with or without interrupt generation).

5.6 How are the product/peripheral specific features managed?

The HAL drivers offer extended APIs, i.e. specific functions as add-ons to the common API to support features available on some products/lines only.

5.7 How can STM32CubeMX generate code based on embedded software?

STM32CubeMX has a built-in knowledge of STM32 microcontrollers, including their peripherals and software, that allows to provide a graphical representation to the user and generate *.h/*.c files based on user configuration.

5.8 How can I get regular updates on the latest STM32CubeL4 firmware releases?

The STM32CubeL4 firmware package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new firmware package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32L4 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeL4 firmware package updates.

For more details, refer to *Section 4.4*.

5.9 When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users.

LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

5.10 How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32l4xx_ll_ppp.h file(s).

5.11 Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers.

The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

5.12 Is there any LL APIs which are not available with HAL

Yes, there are.

A few Cortex® APIs have been added in stm32l4xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

5.13 Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
16-Feb-2015	1	Initial release.
15-Sep-2015	2	Added STM32L475xx and STM32L485xx in *Table 1: Macros for STM32L4 series*. Added Low Layer (LL) drivers in *Section : Introduction*, *Section 1: STM32CubeL4 main features*, *Section 2: STM32CubeL4 architecture overview* and *Section 3: STM32CubeL4 firmware package overview*. Added new Examples_LL and Examples_MIX examples in *Section 3.2: Firmware package overview*/ Added SW4STM32 toolchain in *Section 3.2: Firmware package overview* and in *Section 4.1: Running your first example*. Updated *Section 5: FAQ* to add low-layer drivers.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved